(19) BUNDESREPUBLIK DEUTSCHLAND

DEUTSCHES PATENTAMT

(12) **Offenlegungsschrift**

(10) **DE 44 16 881 A1**

(51) Int. Cl.[5]: G 06 F 15/80

(21) Aktenzeichen: P 44 16 881.0
(22) Anmeldetag: 13. 5. 94
(43) Offenlegungstag: 17. 11. 94

DE 44 16 881 A1

(30) Innere Priorität: (32) (33) (31)
13.05.93 DE 43 16 036.0

(71) Anmelder:
Vorbach, Martin, 76137 Karlsruhe, DE

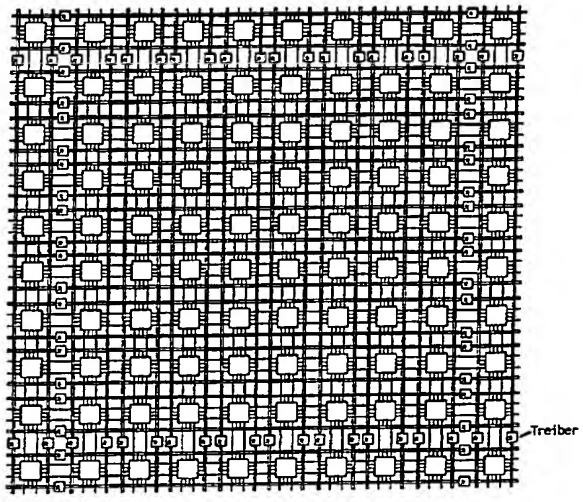
(74) Vertreter:
Zahn, R., Dipl.-Ing., Pat.-Anw., 76229 Karlsruhe

(72) Erfinder:
gleich Anmelder

Prüfungsantrag gem. § 44 PatG ist gestellt

(54) Datenverarbeitungseinrichtung

(57) In Verbindung mit einer Datenverarbeitungseinrichtung, wobei ein (im folgenden Datenflußprozessor - DFP - genannter) integrierter Schaltungskreis (Chip) mit einer Vielzahl insbesondere orthogonal zueinander angeordneter homogen strukturierter Zellen mit je einer Mehrzahl jeweils logisch gleicher und strukturell identisch angeordneter Bausteine vorgesehen ist, dessen Zellen zeilen- und spaltenweise, gegebenenfalls gruppenweise zusammengefaßt, mit Ein-/Ausgangsanschlüssen des integrierten Schaltkreises verbunden sind, ist erfindungsgemäß den Zellen eine Ladelogik zugeordnet, über die sie je für sich und gegebenenfalls gruppenweise zusammengefaßt so programmierbar (konfigurierbar) sind, daß beliebige logische Funktionen und/oder Vernetzungen untereinander verifizierbar sind, und zwar derart, daß eine Manipulation der DFP-Konfiguration während des Betriebes (oder zur Laufzeit), d. h. die Modifikation funktioneller Teile (MACROS) des DFPs erfolgen kann, ohne daß andere funktionelle Teile angehalten werden müssen oder in ihrer Funktion beeinträchtigt werden.



DE 44 16 881 A1

Beschreibung

Die vorliegende Erfindung bezieht sich auf eine Datenverarbeitungseinrichtung, d. h. eine Hardwareeinheit zur logischen Manipulation (Verknüpfung) von in binärer Form vorliegenden Daten (Informationen).

Derartige Datenverarbeitungseinrichtungen sind mittlerweile lange bekannt und sie haben bereits breite Anwendung und Anerkennung gefunden. Die prinzipielle Aufbau- und Arbeitsstruktur der bekannten Datenverarbeitungseinrichtungen ist in etwa so zu definieren, daß eine arithmetisch-logische Verknüpfungseinheit vorgesehen ist, in der die zu verknüpfenden Daten einer programmtechnischen Anweisung (Software) zufolge verarbeitet werden. Die Daten werden dabei über ein Steuerwerk in mehr oder weniger komplexen Adressierungsvorgängen entsprechend abgerufen und zunächst in Arbeitsregistern bereitgestellt; nach der logischen Verknüpfung werden dann die neuen Daten in einer vorgegebenen Speicherstelle wieder abgelegt. Die arithmetisch-logische Verknüpfungseinheit besteht dabei aus logischen Verknüpfungsbausteinen (Gatter, Glieder), die jeweils so miteinander gekoppelt sind, daß die zu manipulierenden Daten der zugrunde liegenden Software entsprechend den vier Grundrechenarten gemäß logisch verarbeitet werden.

Es ist leicht nachzuvollziehen, daß auf der Basis der bekannten Strukturen relativ viel Rechenzeit dafür erforderlich ist, die zu manipulierenden Daten auszulesen, und die Arbeitsregister zu überführen, den spezifischen Logikbausteinen in der arithmetisch-logischen Verknüpfungseinheit zuzuleiten und schließlich wieder abzuspeichern. Es ist ferner einsichtig, daß die Hardware-Struktur der arithmetisch-logischen Verknüpfungseinheit insoweit nicht als optimal betrachtet werden kann, als schließlich die hardwaremäßig vorhandenen integrierten logischen Bausteine stets nur in ein und derselben Art und Weise im Gesamtsystem aktiv benutzt werden. Ebenso wird durch strikte Hardwarevorgabe ein Aneinanderreihen von Funktionen in sogenannten Pipelines sehr erschwert oder eingeschränkt, was zwangsläufig ein häufiges Registerumladen zwischen Arbeitsregistern und Rechenwerk bedeutet. Derartige Bausteine sind des weiteren nur schlecht kaskadierbar und erfordern dann sehr viel Programmierarbeit.

Ein zusätzlicher Vorteil der vorliegenden Erfindung liegt darin, daß eine über einen weiten Raum skalierbare Parallelität zur Verfügung steht. Hierbei wird eine Basis zum schnellen und flexiblen Aufbau von neuronalen Strukturen geschaffen, wie die bis dato lediglich mit erheblichem Aufwand simuliert werden können.

Die der vorliegenden Erfindung zugrunde liegende Aufgabe besteht darin, eine im folgenden Datenflußprozessor (DFP) genannte Datenverarbeitungseinrichtung anzugeben, bei der eine höhere beziehungsweise bessere Flexibilität der Gesamtstruktur und des Datenflusses sowie der Pipelining- und Kaskadiermöglichkeiten zu einer Erhöhung der Rechnerbeziehungsweise Verknüpfungsleistung führt.

Außer dem Einsatz als reiner Datenflußprozessor, soll der DFP folgende weitere Aufgaben erfüllen können:

— Einsatz als universeller Baustein zum Aufbau von herkömmlichen Rechnern, wobei der Aufbau einfacher und billiger werden soll.
— Einsatz in neuronalen Netzen.

Diese Aufgabe wird dadurch gelöst, daß ein integrierter Schaltkreis (Chip) mit einer Vielzahl insbesondere orthogonal zueinander angeordneter Zellen mit je einer Mehrzahl jeweils logisch gleicher und strukturell identisch angeordneter Zellen vorgesehen ist deren Anordnung, sowie die interne Busstruktur, zur Erleichterung der Programmierung äußerst homogen ist. Dennoch ist es denkbar innerhalb eines Datenflußprozessors Zellen mit verschiedenen Zellogiken und Zellstrukturen unterzubringen, um so die Leistungsfähigkeit zu erhöhen, indem zum Beispiel für Speicheransteuerungen andere Zellen als für arithmetische Operationen existieren. Insbesondere kann für neuronale Netze eine gewisse Spezialisierung von Vorteil sein. Den Zellen ist eine Ladelogik zugeordnet, über die die Zellen je für sich und gegebenenfalls gruppenweise in sogenannte MACROs zusammengefaßt so programmierbar sind, daß einerseits beliebige logische Funktionen, andererseits aber auch die Verknüpfung der Zellen untereinander in weiten Bereichen verifizierbar sind. Dies wird erreicht indem jeder einzelnen Zelle ein gewisser Speicherplatz zur Verfügung steht, in dem die Konfigurationsdaten abgelegt sind. Anhand dieser Daten werden Multiplexer oder Transistoren in der Zelle beschaltet um die jeweilige Zellfunktion zu gewährleisten (siehe **Fig. 12**).

Mit anderen als im Patentanspruch 1 gebrauchten Worten besteht der Kern der vorliegenden Erfindung darin, einen Datenflußprozessor vorzuschlagen, der zellular aufgebaut ist und dessen Zellen über eine externe Ladelogik im arithmetisch-logischen Sinne quasi beliebig neu konfiguriert werden können. Dabei ist es von äußerster Notwendigkeit, daß die betreffenden Zeilen einzeln und ohne Beeinflussung der übrigen Zeilen oder gar einer Stillegung des gesamten Bausteins umkonfiguriert werden können. Der Datenflußprozessor gemäß der vorliegenden Erfindung kann so während eines ersten Arbeitszyklus als Addierer und während eines späteren Arbeitszyklus als Multiplizierer "programmiert" werden, wobei die Anzahl der für die Addition beziehungsweise die Multiplikation erforderlichen Zellen durchaus unterschiedlich sein können. Dabei bleibt die Plazierung der bereits geladenen MACROs erhalten; der Ladelogik beziehungsweise dem Compiler obliegt es, das neu zu ladende MACRO innerhalb der freien Zellen zu partitionieren (d. h. das zu ladende MACRO so zu zerlegen, daß es sich optimal einfügen läßt). Die Ablaufsteuerung des Programms wird dabei von der Ladelogik übernommen, indem sie gemäß dem momentan ausgeführten Programmabschnitt die entsprechenden MACROs in den Baustein lädt, wobei der Ladevorgang von der später beschriebenen Synchronisationslogik mitgesteuert wird, indem sie den Zeitpunkt des Umladens festlegt. Daher entspricht der DFP nicht der bekannten von-Neumann-Architektur, da die Daten- und Programmspeicher getrennt sind. Dies bedeutet jedoch gleichzeitig eine höhere Sicherheit, da fehlerhafte Programme keinen CODE, sondern lediglich DATEN zerstören können.

Um den Datenflußprozessor eine arbeitsfähige Struktur zu geben, werden einige Zellen, und zwar unter anderem die Eingabe-/Ausgabefunktionen (I/O) und Speichermanagementfunktionen vor dem Laden der Programme geladen und bleiben für gewöhnlich während der gesamten Laufzeit konstant. Dies ist erforderlich um den Datenflußprozessor an seine Hardwareumgebung anzupassen. Die übrigen Zellen werden zu sogenannten MACROs zusammengefaßt und können während der Laufzeit nahezu beliebig und ohne Beeinflussung der

Nachbarzellen umkonfiguriert werden. Dazu sind die Zellen einzeln und direkt adressierbar.

Um die Umstrukturierung (das Umladen) der Zellen oder MACROs mit der Ladelogik zu synchronisieren, kann — wo notwendig, da nur Umgeladen werden darf, wenn die MACROs mit ihrer alten Tätigkeit fertig sind — eine Synchronisationsschaltung als MACRO auf dem Datenflußprozessor untergebracht werden, die die entsprechenden Signale an die Ladelogik absendet. Hierzu kann eventuell eine Modifikation der gewöhnlichen MACROs von Nöten sein, da diese dann der Synchronisations-Schaltung Zustandsinformationen zur Verfügung stellen müssen.

Diese Zustandsinformationen signalisieren der Synchronisationslogik für gewöhnlich, daß einzelne MACROs ihre Aufgabe erledigt haben, was aus programmiertechnischer Sicht zum Beispiel die Terminierung einer Prozedur oder das Erreichen der Terminierungsbedingung einer Schleife bedeuten kann. D.h. das Programm wird an einer anderen Stelle fortgesetzt und die die Zustandsinformation absendenden MACROs können umgeladen werden. Zudem kann es von Interesse sein, daß die MACROs in einer bestimmten Reihenfolge umgeladen werden. Hierzu kann eine Wertung der einzelnen Synchronisations-Signale durch einen Prioritätsdekoder erfolgen. Eine derartige — einfache — Logik ist in **Fig.** 13 gezeichnet. Die Logik besitzt sieben Eingangssignale durch die die sieben MACROs ihre Zustandsinformation abgeben. In diesem Fall soll 0 für "in Arbeit" und 1 für "fertig" stehen. Die Logik besitzt drei Ausgangssignale, die an die Ladelogik geführt werden, wobei der Zustand 000 als Ruhezustand gilt. Liegt ein Signal an einem der sieben Eingänge an, so findet eine Dezimal-Binär-Umsetzung statt, so wird zum Beispiel Sync6 als 110 dargestellt, was der Ladelogik anzeigt, daß das MACRO, welches Sync6 bedient, seine Aufgabe beendet hat. Liegen gleichzeitig mehrere Synchronisations-Signale am Eingang an, so gibt die Synchronisationsschaltung das Signal mit der höchsten Priorität an die Ladelogik weiter; liegen zum Beispiel Sync0, Sync4 und Sync6 an, so reicht die Syncronistaions-Schaltung zunächst Sync6 an die Ladelogik weiter. Nachdem die entsprechenden MACROs umgeladen sind und somit Sync6 nicht mehr anliegt wird Sync4 weitergeleitet usw. Zur Verdeutlichung dieses Prinzips kann der Standard-TTL-Baustein 74148 in Betracht gezogen werden.

Über die Ladelogik kann der Datenflußprozessor jeweils optimal und gegebenenfalls dynamisch auf eine zu lösende Aufgabe eingestellt werden. Damit ist zum Beispiel der große Vorteil verbunden, daß neue Normen oder dergleichen einzig und allein durch eine Umprogrammierung des Datenflußprozessors umgesetzt werden können und nicht — wie bisher — einen Austausch mit entsprechendem Anfall von Elektronikschrott bedingen.

Die Datenflußprozessoren sind untereinander kaskadierbar, was zu einer beinahe beliebigen Erhöhung des Parallelisierungsgrades, der Rechenleistung, sowie der Netzgröße in neuronalen Netzen führt. Besonders wichtig ist hier eine klare homogene Verbindung der Zellen mit den Ein-/Ausgangs-Pins (IO-Pin) der Datenflußprozessoren, um möglichst keine Einschränkungen auf die Programme zu haben.

In **Fig.** 14 ist zum Beispiel die Kaskadierung von vier DFPs gezeigt. Sie erscheinen der Umgebung wie ein großer homogener Baustein (**Fig.** 15). Prinzipiell sind damit zwei Kaskadierungsmethoden denkbar:

a) Nur die lokalen Verbindungen zwischen den Zellen werden herausgeführt, was im vorliegenden Beispiel zwei IO-Pins pro Kantenzelle und vier IO-Pins pro Eckzelle bedeutet. Allerdings hat der Compiler/Programmierer zu beachten, daß die globalen Verbindungen nicht herausgeführt werden, wodurch die Kaskadierung nicht vollständig homogen ist. (Globale Verbindungen zwischen mehreren Zellen, für gewöhnlich zwischen einer kompletten Zellenreihe oder -spalte — siehe **Fig.** 6 —; lokale Verbindungen existieren nur zwischen zwei Zellen). **Fig.** 16a zeigt den Aufbau innerhalb eines DFPs, **Fig.** 17a zeigt die daraus resultierende Kaskadierung von mehreren DFPs (drei gezeichnet).

b) Die lokalen und globalen Verbindungen werden herausgeführt, was die Anzahl der benötigten Treiber/IO-Pins und Leitungen drastisch erhöht, in unserem Beispiel auf sechs IO-Pins pro Kantenzelle und zwölf IO-Pins pro Eckzelle. Dadurch ist eine vollständige Homogenität bei der Kaskadierung gegeben.

Da die globalen Verbindungen insbesondere bei Verwendung der Kaskadierungstechnik b) sehr lang werden können, kann der unangenehme Effekt auftreten, daß die Zahl der globalen Verbindungen nicht ausreicht, da bekanntlich jede Verbindung nur von einem Signal genutzt werden kann. Um diesen Effekt zu minimieren, kann nach einer gewissen Länge der globalen Verbindungen ein Treiber eingeschleift werden. Der Treiber hat zum einen eine Verstärkung des Signals zur Aufgabe, die bei langen Strecken und entsprechend hohen Lasten, unbedingt erforderlich ist; zum anderen kann der Treiber in Tristate gehen und damit das Signal unterbrechen. Dadurch können die Abschnitte links und rechts, beziehungsweise oberhalb und unterhalb des Treibers von verschiedenen Signalen genutzt werden, sofern der Treiber in Tristate ist, ansonsten wird ein Signal durchgeschleift. Wichtig ist hierbei, daß die Treiber der einzelnen globalen Leitungen auch einzeln angesteuert werden können, d. h. ein globales Signal kann unterbrochen sein, das Nachbarsignal ist jedoch durchgeschleift. Somit können auf einer globalen Verbindung durchaus abschnittweise verschiedene Signale anliegen, während die globale Nachbarverbindung tatsächlich global von ein und demselben Signal verwendet wird (vergleiche **Fig.** 22).

Zur besseren Kommunikation zwischen den Datenflußprozessoren und der Ladelogik können sogenannte Shared-Memories eingesetzt werden. So können zum Beispiel Programme von einer Festplatte, die im IO-Bereich eines Datenflußprozessors liegt zur Ladelogik durchgereicht werden, indem die Datenflußprozessoren die Daten von der Platte in den Shared-Memory schreiben und die Ladelogik sie dort abholt. Dies ist besonders wichtig, da hier, wie bereits erwähnt, keine von-Neumann- sondern eine Harvardarchitektur vorliegt. Ebenso sind die Shared-Memories von Vorteil, wenn Konstanten, die im Programm — das im Speicherbereich der Ladelogik liegt — definiert sind, mit Daten — die im Speicherbetrieb der Datenflußprozessoren liegen — verknüpft werden sollen.

Weiterbildungen der vorstehend definierten und umschriebenen Erfindung sind Gegenstand der Unteransprüche.

Eine besondere Verwendung des erfindungsgemäßen Datenflußprozessors ist darin zu sehen, daß er in Verbindung mit geeigneten Ein-/Ausgabe-Einheiten einer-

seits und einem Speicher andererseits die Basis für einen kompletten (komplexen) Rechner bilden kann. Dabei kann ein Großteil der IO-Funktionen als MACROs auf dem Datenflußprozessor implementiert werden und es brauchen momentan lediglich Spezialbausteine (Ethernet-Treiber, VRAMS ...) extern zugefügt zu werden. Bei einer Normänderung oder Verbesserung muß dann wie bereits angedeutet nur das MACRO softwareseitig gewechselt werden; ein Eingriff in die Hardware ist nicht notwendig. Es bietet sich hier an, einen IO-(Eingabe-/Ausgabe-) Stecker festzulegen, über welchen dann die Zusatzbausteine angeschlossen werden können.

Fig. 20 zeigt den stark vereinfachten Aufbau eines heute üblichen Rechners. Durch den Einsatz eines DFP-Bausteins können erhebliche Teile eingespart werden (Fig. 21), wobei die entsprechenden herkömmlichen Baugruppen (CPU, Speicherverwaltung, SCSI-, Tastatur- und Videointerface, sowie der parallelen und seriellen Schnittstellen) als MACROs in die kaskadierten DFPs abgelegt werden. Nur die durch einen DFP nicht nachbildbaren Teile wie Speicher und Leitungstreiber mit nicht TTL-Pegeln oder für hohe Lasten müssen extern zugeschaltet werden. Durch die Verwendung des DFPs ist eine günstige Produktion gegeben, da ein und derselbe Baustein sehr häufig verwendet wird, das Layout der Platine ist durch die homogene Vernetzung entsprechend einfach. Zudem wird der Aufbau des Rechners durch die Ladelogik bestimmt, die hier für gewöhnlich nur zu Beginn der Abarbeitung (nach einem Reset) das DFP-Array lädt, wodurch eine günstige Fehlerkorrektur- und Erweiterungsmöglichkeit gegeben ist. Ein derartiger Rechner kann insbesondere mehrere verschiedene Rechnerstrukturen simulieren, indem einfach der Aufbau des zu simulierenden Rechners in das DFP-Array geladen wird. Zu bemerken ist, daß hierbei der DFP nicht in seiner Funktion als DFP arbeitet sondern lediglich ein hochkomplexes und frei programmierbares Logikarray zur Verfügung stellt, sich hierbei jedoch von herkömmlichen Bausteinen in seiner besonderen guten Kaskadierbarkeit unterscheidet.

Ein weiteres Einsatzgebiet des Bausteins ist der Aufbau großer neuronaler Netze. Sein besonderer Vorzug liegt hierbei in seiner hohen Gatterdichte, seiner ausgezeichneten Kaskadierbarkeit, sowie seiner Homogenität. Ein Lernvorgang, der eine Änderung einzelner axiomatischer Verbindungen beziehungsweise einzelner Zellfunktionen beinhaltet ist auf üblichen Bausteinen ebenso schlecht durchführbar, wie der Aufbau großer homogener und gleichzeitig flexibler Zellstrukturen. Die dynamische Umkonfigurierbarkeit ermöglicht erstmalig die optimale Simulation von Lernvorgängen.

Die vorliegende Erfindung wird im folgenden anhand der weiteren Figuren näher erläutert. Insgesamt zeigen

Fig. 1 ein Schaltsymbol für einen 8-Bit-Addierer;

Fig. 2 ein Schaltsymbol für einen aus acht 1-Bit-Addierern bestehenden 8-Bit-Addierer nach Fig. 1;

Fig. 3 eine logische Struktur eines 1-Bit-Addierers entsprechend Fig. 2;

Fig. 4 eine Zellenstruktur des 1-Bit-Addierers entsprechend Fig. 3;

Fig. 5 einen der Zellenstruktur nach Fig. 1 entsprechend aufgebauten 8-Bit-Addierer;

Fig. 6 ein aus vier Zellen bestehendes unprogrammiertes SUBMACRO X (analog einem 1-Bit-Addierer gemäß Fig. 4 beziehungsweise Fig. 5) mit den erforderlichen Leitungsanschlüssen;

Fig. 7 einen Teilausschnitt eines integrierten Schaltkreises (Chip) mit einer Vielzahl von Zellen und einem separierten SUBMACRO X gemäß Fig. 6;

Fig. 8 einen integrierten Schaltkreis (Chip) mit einer Orthogonalstruktur einer quasi beliebigen Vielzahl von Zellen und einer extern zugeordneten Ladelogik;

Fig. 9 ein erstes Ausführungsbeispiel einer Mehrzahl miteinander zu einem Rechenwerk gekoppelter integrierter Schaltkreise (Datenflußprozessor) nach Fig. 8;

Fig. 10 ein zweites Ausführungsbeispiel einer Mehrzahl miteinander zu einem Rechenwerk gekoppelter integrierter Schaltkreise (Datenflußprozessor) nach Fig. 8;

Fig. 11 ein Ausführungsbeispiel eines MACRO zur Addition zweier Zahlenreihen;

Fig. 12 einen beispielhaften Aufbau einer Zelle mit Multiplexern zur Auswahl der jeweiligen logischen Bausteine;

Fig. 13 eine zum Beispiel mit einem Standard-TTL-Baustein 74148 ausgeführte Synchronisationslogik;

Fig. 14 die Kaskadierung von vier DFPs, wobei die Verbindung zwischen den IO-Pins nur schematisch dargestellt sind (tatsächlich bedeutet eine gezeichnete Verbindung eine Mehrzahl von Leitungen);

Fig. 15 die durch die Kaskadierung erreichte Homogenität;

Fig. 16a die Struktur der E/A-Zellen, wobei die globalen Verbindungen nicht herausgeführt werden,

Fig. 16b die Struktur der E/A-Zellen, jedoch mit herausgeführten globalen Verbindungen;

Fig. 17a die aus Fig. 16a resultierende Kaskadierung, wobei eine Eckzelle, sowie die zwei mit ihr kommunizierenden Treiberzellen der kaskadierten Bausteine (vergleiche hierzu Fig. 14) gezeichnet sind;

Fig. 17b die aus Fig. 16b resultierende Kaskadierung, wobei eine Eckzelle, sowie die zwei mit ihr kommunizierenden Treiberzellen der kaskadierten Bausteine (vergleiche hierzu Fig. 14) gezeichnet sind;

Fig. 18a eine Multiplikationsschaltung (vergleiche Fig. 11a);

Fig. 18b die interne Struktur des DFPs nach dem Laden (vergleiche Fig. 11b);

Fig. 19c die Arbeitsweise des DFPs im Speicher, sowie die Zustände der Zähler 47, 49;

Fig. 19 eine Kaskadenschaltung, wobei der Addierer aus Fig. 11 und der Multiplizierer aus Fig. 18 zur Steigerung der Rechenleistung hintereinander geschaltet sind;

Fig. 20 den stark schematisierten Aufbau eines herkömmlichen Rechners;

Fig. 21 den möglichen Aufbau desselben Rechners mit Hilfe eines Arrays aus kaskadierten DFPs;

Fig. 22 einen Ausschnitt mit eingezeichneten (Leitungs-) Treibern eines DFPs.

In Fig. 1 ist ein Schaltsymbol eines 8-Bit-Addierers dargestellt. Das Schaltsymbol besteht aus einem quadratischen Baustein 1 mit acht Eingängen A 0 ... 7 für ein erstes Datenwort A und acht Eingängen B 0 ... 7 für ein zweites (zu addierendes) Datenwort B. Die jeweils acht Eingänge Ai, Bi werden ergänzt durch einen weiteren Eingang Üein über den dem Baustein 1 gegebenenfalls ein Übertrag zugeleitet wird. Der Baustein 1 hat funktions- und bestimmungsgemäß acht Ausgänge S 0 ... 7 für binären Summanden und einen weiteren Ausgang Üaus für den gegebenenfalls bestehenden Übertrag.

Das in Fig. 1 dargestellte Schaltsymbol ist in Fig. 2 als Anordnung sogenannter SUBMACROS dargestellt. Diese SUBMACROS 2 bestehen je aus einem 1-Bit-Addierer 3 mit je einem Eingang für die entsprechenden Bits des Datenworts und einem weiteren Eingang für ein

Übertragsbit. Die 1-Bit-Addierer 3 weisen darüberhinaus einen Ausgang für den Summanden und einen Ausgang für den Übertrag Üaus auf.

In **Fig.** 3 ist die binäre Logik eines 1-Bit-Addierers beziehungsweise eines SUBMACROS **2** nach **Fig.** 2 dargestellt. Analog zu **Fig.** 2 weist diese Schaltlogik je einen Eingang Ai, Bi für die konjugierten Bits der zu verknüpfenden Daten auf; ferner ist ein Eingang Üein für den Übertrag vorgesehen. Diese Bits werden den dargestellten Verbindungen beziehungsweise Verknüpfungen entsprechend in zwei ODER-Gliedern **5** und drei NAND-Gliedern **6** verknüpft, so daß am Ausgangsanschluß Si und am Ausgang für den Übertrag Üaus die einem Volladdierer entsprechenden Verknüpfungsergebnisse (Si, Üaus) anstehen.

Die Erfindung setzt da ein, wo es — wie in **Fig.** 4 dargestellt — darum geht, das in **Fig.** 3 gezeigte SUBMACRO **2** oder eine oder mehrere beliebige Funktion(en) in geeigneter Weise in einer Zellstruktur zu implementieren. Dies geschieht auf der Grundlage logisch und strukturell identischer Zellen **10**, deren einzelne logische Bausteine der auszuführenden Verknüpfungsfunktion entsprechend miteinander gekoppelt werden, und zwar mittels der noch zu beschreibenden Ladelogik. Gemäß der in **Fig.** 4 gezeigten, von der Schaltlogik nach **Fig.** 3 abgeleiteten Verknüpfungslogik für einen 1-Bit-Addierer sind je zwei Zellen **10.1**, **10.2** bezüglich der logischen Bausteine insoweit gleich, daß jeweils ein ODER-Glied **5** und ein NAND-Glied **6** aktiviert sind. Eine dritte Zelle **10.3** wird nur als Leitungszelle (Leiterbahnzelle) benutzt und die vierte Zelle **10.4** ist bezüglich des dritten NAND-Gliedes **6** aktiv geschaltet. Das aus den vier Zellen **10.1** ... **10.4** bestehende SUBMACRO **2** steht somit stellvertretend für einen 1-Bit-Addierer, d. h. ein 1-Bit-Addierer einer Datenverarbeitungseinrichtung gemäß der vorliegenden Erfindung wird über vier entsprechend programmierte (konfigurierte) Zellen **10.1** ... **10.4** verifiziert. (Der Vollständigkeit halber soll angemerkt werden, daß die einzelnen Zellen ein erheblich umfangreicheres Netzwerk von logischen Bausteinen, sprich Verknüpfungsgliedern, und Invertern aufweist, die jeweils dem aktuellen Befehl der Ladelogik zufolge aktiv geschaltet werden können. Neben den logischen Bausteinen ist auch ein dichtes Netz von Verbindungsleitungen zwischen den jeweils benachbarten Bausteinen und zum Aufbau von zeilen- und spaltenweisen Busstrukturen zur Datenübertragung andererseits vorgesehen, so daß über eine entsprechende Programmierung seitens der Ladelogik quasi beliebige logische Verknüpfungsstrukturen implementiert werden können).

Der Vollständigkeit halber ist in **Fig.** 5 der Zellenaufbau eines 8-Bit-Addierers in seiner Gesamtheit dargestellt. Die in **Fig.** 5 gezeigte Struktur entspricht insoweit der nach **Fig.** 2, wobei die in **Fig.** 2 symbolisch als SUBMACROS **2** dargestellten 1-Bit-Addierer jeweils durch eine vier-zellige Einheit **10.1** ... **10.4** ersetzt sind. Bezogen auf den erfindungsgemäßen Datenflußprozessor bedeutet dies, daß zweiunddreißig Zellen der zur Verfügung stehenden Gesamtheit von Zellen einer zellular mit logisch identischem Layout gefertigten Schaltungsplatine seitens der Ladelogik so angesteuert und konfiguriert beziehungsweise programmiert werden, daß diese zweiunddreißig Zellen ein 8-Bit-Addierer bilden.

In der Darstellung nach **Fig.** 5 ist über eine strichpunktierte Umrahmung ein SUBMACRO "X" zeichnerisch separiert, das letztlich als aus vier einem 1-Bit-Addierer entsprechend programmierten Zellen (**10** gemäß **Fig.** 4) bestehende Untereinheit zu betrachten ist.

Das in **Fig.** 5 separierte SUBMACRO "X" ist in **Fig.** 6 als Teil eines integrierten Schaltkreises (Chip) **20** gemeinsam mit Leitungs- und Datenanschlüssen dargestellt. Das SUBMACRO "X" besteht aus den vier Zellen **10** die entsprechend der orthogonalen Struktur je Seite vier Datenanschlüsse (also insgesamt sechzehn Datenanschlüsse je Zelle) aufweisen. Die Datenanschlüsse verbinden jeweils benachbarte Zellen, so daß ersichtlich wird, wie beispielsweise eine Dateneinheit von Zelle zu Zelle durchgeschleust wird. Die Ansteuerung der Zellen **10** erfolgt einerseits über sogenannte lokale Steuerungen, das sind lokale Leitungen, die mit allen Zellen verbunden sind, und andererseits über sogenannte globale Leitungen, d. h. Leitungen, die über den gesamten integrierten Schaltkreis (Chip) **20** geführt sind.

In **Fig.** 7 ist ein vergrößerter Ausschnitt eines integrierten Schaltkreises **20** dargestellt, der mit einem orthogonalen Raster von Zellen **10** belegt ist. Wie in **Fig.** 7 angedeutet kann so zum Beispiel eine Gruppe von vier Zellen **10** als SUBMACRO "X" ausgewählt und dem 1-Bit-Addierer entsprechend **Fig.** 4 gemäß programmiert beziehungsweise konfiguriert werden.

Ein vollständiger integrierter Schaltkreis (Chip) **20** ist in **Fig.** 8 dargestellt. Dieser integrierte Schaltkreis **20** besteht aus einer Vielzahl im orthogonalen Raster angeordneter Zellen **10** und weist an seinen Außenkanten eine entsprechende Anzahl von Leitungsanschlüssen (Pins) auf, über die Signale, insbesondere Ansteuersignale und Daten zugeführt und weitergeleitet werden können. In **Fig.** 8 ist wiederum das SUBMACRO "X" gemäß **Fig.** 5/**Fig.** 6 abgegrenzt; darüberhinaus sind auch weitere SUBMACROS separiert, die spezifischen Funktionen und Vernetzungen entsprechend zu Untereinheiten zusammengefaßt sind. Dem integrierten Schaltkreis (Chip) **20** ist eine Ladelogik **30** zugeordnet beziehungsweise übergeordnet, über die der integrierte Schaltkreis **20** programmiert und konfiguriert wird. Die Ladelogik **30** teilt letztlich dem integrierten Schaltkreis **20** mit, wie er arithmetisch-logisch zu arbeiten hat. Bezugnehmend auf die **Fig.** 1 bis 5 ist in **Fig.** 8 einerseits das SUBMACRO "X" entsprechend **Fig.** 4 und **Fig.** 5 hervorgehoben; andererseits ist auch ein MACRO "Y" entsprechend **Fig.** 1 und **Fig.** 2 angezeichnet, das als Einheit einem 8-Bit-Addierer entspricht.

Anhand von **Fig.** 9 beziehungsweise **Fig.** 10 soll im folgenden eine Rechnerstruktur beschrieben werden, die auf den im vorstehenden definierten und erläuterten integrierten Schaltkreis **20** aufbaut.

Gemäß dem in **Fig.** 9 dargestellten ersten Ausführungsbeispiel ist — analog zur Anordnung der Zellen — im Orthogonalraster eine Mehrzahl von integrierten Schaltkreisen **20** angeordnet, deren jeweils benachbarte über lokale BUS-Leitungen **21** miteinander gekoppelt beziehungsweise vernetzt sind. Die — beispielsweise aus sechzehn integrierten Schaltkreisen **20** bestehende — Rechnerstruktur weist Ein-/Ausgangsleitungen IO auf, über die der Rechner quasi mit der Außenwelt in Verbindung steht, d. h. korrespondiert. Der Rechner gemäß **Fig.** 9 weist ferner einen Speicher **22** auf, der dem dargestellten Ausführungsbeispiel entsprechend aus zwei separierten Speichern, zusammengesetzt aus jeweils RAM, ROM sowie einem Dual-Ported RAM als shared memory zu der Ladelogik geschaltet, besteht, die gleichermaßen als Schreib-Lese-Speicher oder auch nur als Lese-Speicher realisiert sein können. Der soweit beschriebenen Rechnerstruktur ist die Ladelogik **30** zubeziehungsweise übergeordnet, mittels der die inte-

grierten Schaltkreise (Datenflußprozessor) 20 programmiert und konfiguriert und vernetzt werden.

Die Ladelogik 30 baut auf einem Transputer 31, d. h. einem Prozessor mit mikrocodiertem Befehlssatz auf, dem seinerseits ein Speicher 32 zugeordnet ist. Die Verbindung zwischen dem Transputer 31 und dem Datenflußprozessor basiert auf einer Schnittstelle 33 für die sogenannten Ladedaten, d. h. die Daten die den Datenflußprozessor aufgabenspezifisch programmieren und konfigurieren und einer Schnittstelle 34 für den bereits genannten Rechnerspeicher 22, d. h. den Shared-Memory-Speicher.

Die in Fig. 9 dargestellte Struktur stellt so einen kompletten Rechner dar, der über die Ladelogik 30 jeweils fall- beziehungsweise aufgabenspezifisch programmiert und konfiguriert werden kann. Der Vollständigkeit halber sei noch angemerkt, daß — wie in Verbindung mit der Ladelogik 30 über Pfeile angedeutet — mehrere dieser Rechner vernetzt, d. h. miteinander gekoppelt werden können.

Ein weiteres Ausführungsbeispiel einer Rechnerstruktur ist in Fig. 10 dargestellt. Im Unterschied zu Fig. 9 sind dabei neben den lokalen BUS-Leitungen zwischen den benachbarten integrierten Schaltkreisen 20 noch übergeordnete zentrale BUS-Leitungen 23 vorgesehen, um zum Beispiel spezifische Ein- beziehungsweise Ausgangsprobleme lösen zu können. Auch der Speicher 22 (Shared-Memory) ist über zentrale BUS-Leitungen 23 mit den integrierten Schaltkreisen 20 verbunden, und zwar wie dargestellt jeweils mit Gruppen dieser integrierten Schaltkreise. Die in Fig. 10 dargestellte Rechnerstruktur weist die gleiche Ladelogik 30 auf, wie sie anhand von Fig. 9 erläutert wurde.

In Verbindung mit Fig. 11a soll eine auf erfindungsgemäßen Datenflußprozessoren aufgebaute Additionsschaltung erläutert werden. Ausgegangen wird von zwei Zahlenreihen An und Bn für sämtliche n zwischen 0 und 9; die Aufgabe besteht darin, die Summe $Ci = Ai + Bi$ zu bilden, wobei der Index i die Werte $0 \leq n < 9$ annehmen kann.

Bezugnehmend auf die Darstellung nach Fig. 11a ist die Zahlenreihe An in einem ersten Speicher RAM1 abgespeichert und zwar zum Beispiel ab einer Speicheradresse 1000h; die Zahlenreihe Bn ist in einem Speicher RAM2 an einer Speicheradresse 0dfa0h abgespeichert; die Summe Cn wird in den RAM1 eingeschrieben und zwar unter der Adresse 100ah.

Es ist ein weiterer Zähler 49 zugeschaltet, der lediglich die einzelnen durch die Steuerschaltung freigegebenen Taktzyklen hochzählt. Dies soll im Weiteren zur Verdeutlichung der Umkonfigurierbarkeit einzelner MACROs ohne Beeinflussung der an der Umkonfigurierung nicht beteiligten MACROs dienen.

Fig. 11a zeigt zunächst die eigentliche Additionsschaltung 40, die aus einem ersten Register 41 zur Aufnahme der Zahlenreihe An und einem zweiten Register 42 zur Aufnahme der Zahlenreihe Bn besteht. Den beiden Registern 41/42 ist ein 8-Bit-Addierer entsprechend dem in Fig. 1 dargestellten MACRO 1 nachgeschaltet. Der Ausgang des MACRO 1 führt über eine Treiberschaltung 43 zurück zum Speicher RAM1. Die Taktbeziehungsweise Zeitsteuerung der Additionsschaltung 40 erfolgt über eine von einem Taktgenerator T angesteuerte Zeitsteuerung (STATEMACHINE) 45, die mit den Registern 41, 42 und der Treiberschaltung 43 verbunden ist.

Die Additionsschaltung 40 wird funktional durch eine Adreßschaltung 46 zur Generierung der Adreßdaten für die abzuspeichernden Additionsergebnisse ergänzt. Die Adreßschaltung 46 besteht ihrerseits aus drei MACROs 1 (gemäß Fig. 1) zur Bildung der Adreßdaten, wobei diese MACROs 1 wie folgt geschaltet sind: Über jeweils einen Eingang werden die zu verknüpfenden Adressen für An, Bn, Cn zugeführt. Diese Adressen werden mit den Ausgangssignalen eines Zählers 47 addiert und mit der Statemachine 45 so verknüpft, daß am Ausgang die neue Zieladresse ansteht. Der Zähler 47 und der Komparator 48 haben dabei die Aufgabe sicherzustellen, daß jeweils die richtigen Summanden verknüpft werden und daß jeweils am Ende der Zahlenreihen, d. h. bei $n = 9$ abgebrochen wird. Ist die Addition vollendet, so wird in der Zeitsteuerung 45 ein STOP-Signal generiert und die Schaltung passiv geschaltet. Ebenso kann das STOP-Signal als Eingangssignal für eine Synchronisations-Schaltung verwendet werden, indem die Synchronisationslogik anhand dieses Signals erkennen kann, daß die Gesamtfunktion "Addieren" gemäß dem nachfolgend beschriebenen ML1 Programm beendet ist und die MACROs somit durch neue ersetzt werden können (zum Beispiel könnte STOP das Signal Sync5 sein).

Der Zeitablauf in der Zeitsteuerung 45 (STATEMACHINE) läßt sich dabei wie folgt darstellen, wobei noch anzumerken ist, daß in der Zeitsteuerung 45 eine Verzögerungszeit T (in Form von Taktzyklen) zwischen der Adreßgenerierung und dem Datenerhalt implementiert ist:

- Im Zyklus 1 wird jeweils der Zähler 47 um 1 erhöht und im Komparator 48 wird geprüft, ob $n > 9$ erreicht ist; synchron zu diesen Operationen werden die Adressen für A, B, C berechnet;
- im Zyklus $(T + 1)$ werden die Summanden A, B ausgelesen und addiert;
- im Zyklus $(T + 2)$ wird die Summe C abgespeichert.

Mit anderen Worten heißt dies, daß die Operationsschleife und die eigentliche Addition gerade $(T + 2)$ Taktzyklen erfordert. Im allgemeinen sind für T 2 ... 3 Takte erforderlich, so daß verglichen mit den herkömmlichen Prozessoren (CPU), die im allgemeinen 50 bis mehrere 100 Taktzyklen bedingen, eine ganz wesentliche Rechenzeit-Reduzierung möglich wird.

Die anhand von Fig. 11 aufgezeigte Konfiguration soll im folgenden über eine hypothetische MACRO-Sprache ML1 nochmals erläutert werden:

Es existieren die Zahlenreihen An und Bn

$\forall n: 0 \leq n \leq 9$

Es sollen die Summen $Ci = Ai + Bi$ mit $I \in N$ gebildet werden.

```
const n = 9;
array A[n] in RAM[1] at 1000h;
array B[n] in RAM[2] at 0dfa0h;
array C[n] in RAM[1] at 100ah;
for i = 0 to n with (A[i], B[i], C[i])
Δ1;
C = Δ1 = A + B;
next;
```

RAM1 ist der 1. Speicherblock
RAM2 ist der 2. Speicherblock
at folgt die Basisadresse der Arrays
for ist der Schleifenbeginn
next ist das Schleifenende
with ( ) folgen die Variablen, deren Adressen durch die

Zählvariable i bestimmt werden
Δ T folgt die Verzögerungszeit für eine Statemachine in Taktzyklen.

Das Timing der Statemachine sieht demnach folgendermaßen aus:

Zyklus	Aktivität
1	Zähler erhöhen, Vergleich auf > 9 (ja ⇒ Abbruch) und Adressen für A, B, C, berechnen
T + 1	A, B, holen und addieren
T + 2	Nach C speichern

Das heißt — wie bereits erwähnt — die Schleife und die Addition benötigen gerade einmal T + 2 Taktzyklen.

Fig. 11b zeigt den groben Aufbau der einzelnen Funktionen (MACROs) in einem DFP. Die MACROs sind in ihrer etwaigen Lage und Größe eingezeichnet und mit den anhand von **Fig.** 11a erläuterten entsprechenden Nummern versehen.

Fig. 11c zeigt den groben Aufbau der einzelnen Funktionen auf die RAM-Blöcke 1 und 2: Die Summanden werden nacheinander in aufsteigender Reihenfolge aus den RAM-Blöcken 1 und 2 ab Adresse 1000h beziehungsweise 0dfa0h gelesen und in RAM-Block 1 ab Adresse 100ah gespeichert. Zudem sind die Zähler **47** und **49** gegeben, beide zählen während des Ablaufs der Schaltung von 0 bis 9.

Nach Beendigung des beschriebenen Programms soll ein neues Programm geladen werden, das die Ergebnisse weiterverwertet. Die Umladung soll zur Laufzeit erfolgen. Das Programm ist im Folgenden gegeben:

Es existieren die Zahlenreihen An und Bn, wobei An durch das Ergebnis Cn des vorher ausgeführten Programms gegeben ist:

$\forall n: 0 \leq n \leq 9$

Es sollen die Produkte $Ci = Ai \cdot Bi$ mit $I \in N$ gebildet werden.

```
const n=9
array A[n] in RAM[1] at 100ah
array B[n] in RAM[2] at 0dfa0h
array C[n] in RAM[1] at 1015h
for i=0 to n with (A[i], B[i], c[i])
Δ1;
C= Δ1=A • B;
next;
```

Die Beschreibung der einzelnen Befehle ist bereits bekannt,
• symbolisiert die Multiplikation.

Die MACRO-Struktur ist in **Fig.** 18a beschrieben, **Fig.** 18b gibt in bekannter Weise die Lage und Größe der einzelnen MACROs auf dem Chip an, besonders zu beachten ist die Größe des Multiplizierers 2 in Vergleich zu Addierer 1 aus **Fig.** 11b. In **Fig.** 18c ist erneut die Auswirkung der Funktion auf den Speicher aufgezeigt, Zähler **47** zählt erneut von 0 bis 9, d. h. er wird beim Nachladen der MACROs zurückgesetzt.

Besonders zu beachten ist der Zähler **49**. Angenommen, das Umladen der MACROs beträgt 10 Taktzyklen. Dann läuft der Zähler **49** von 9 auf 19, da der Baustein dynamisch umgeladen wird, d. h. nur die umzuladenden Teile werden gestoppt, der Rest arbeitet weiter. Das führt nun dazu, daß der Zähler während des Programmablaufs von 19 auf 29 hochläuft. (Hiermit soll das dynamische unabhängige Umladen demonstriert werden, in jedem bisher bekannten Baustein würde der Zähler erneut von 0 auf 9 laufen, da er zurückgesetzt wird).

Bei näherer Betrachtung des Problems stellt sich die Frage, warum nicht beide Operationen, die Addition und die Multiplikation in einem Zyklus durchgeführt werden, also die Operation:

Es existieren die Zahlenreihen An und Bn, wobei An durch das Ergebnis von Cn des vorher ausgeführten Programms gegeben ist:

$\forall n: 0 \leq n \leq 9$

Es sollen die Produkte $Ci = (Ai + Bi) \cdot Bi$ mit $I \in N$ gebildet werden.

```
path D;
const n=9;
array A[n] in RAM[1] at 1000h
array B[n] in RAM[2] at 0dfa0h
array C[n] in RAM[1] at 100ah
for i=0 to n with (A[i], B[i], C[i])
Δ1;
D=Δ1=AbB;
C=Δ1=D • B;
next;
```

path D definiert einen internen nicht aus den DFP herausgeführten Doppelpfad. Die Operation benötigt wegen einem zusätzlichen Δ1 einen Taktzyklus mehr als vorher, ist insgesamt jedoch schneller als die beiden obigen Programme in Folge ausgeführt, da zum einen die Schleife nur einmal durchlaufen wird, zum zweiten nicht umgeladen wird.

Prinzipiell könnte das Programm auch so formuliert werden:

```
const n=9;
array A[n] in RAM[1] at 1000h
array B[n] in RAM[2] at 0dfa0h
array C[n] in RAM[i] at 100ah
for i=0 to n with (A[i], B[i], C[i])
Δ— 1;
C=Δ2=(A+B) • B;
next;
```

Sind die Gatterlaufzeiten des Addierers und des Multiplizierers zusammen kleiner als ein Taktzyklus, kann die Operation (A + B) • B auch in einem Taktzyklus durchgeführt werden, was zu einer weiteren erheblichen Geschwindigkeitssteigerung führt:

```
const n=9;
array A[n] in RAM[1] at 1000h
array B[n] in RAM[2] at Odfa0h
array C[n] in RAM[1] at 100ah
for i=0 to n with (A[i], B[i], C[i])
Δ1;
C=Δ1=(A+B) • B;
next;
```

Anhand von **Fig.** 12 soll ein einfaches Beispiel eines Zellenaufbaus erläutert werden. Die Zelle **10** umfaßt zum Beispiel ein UND-Glied **51**, ein ODER-Glied **52**, ein XOR-Glied **53**, einen Inverter **54** sowie eine Registerzelle **55**. Die Zelle **10** weist darüberhinaus eingangsseitig zwei Multiplexer **56, 57** mit (den sechzehn Eingängen der Zelle entsprechend **Fig.** 6) zum Beispiel je sechzehn Eingangsanschlüssen IN1, IN2 auf. Über diesen (16 : 1)-Multiplexer **56/57** werden jeweils die den genannten logischen Gliedern UND, ODER, XOR **51** ... **53** zuzuführenden Daten ausgewählt. Diese logischen Glieder sind ausgangsseitig mit einem (3 : 1)-Multiple-

xer 58 gekoppelt, der seinerseits mit dem Eingang des Inverters 54, einem Eingang der Registerzelle 55 und einem weiteren (3 : 16)-Multiplexer 59 gekoppelt ist. Der letztgenannte Multiplexer 59 ist zusätzlich mit dem Ausgang des Inverters 54 und einem Ausgang der Registerzelle 55 verbunden und gibt das Ausgangssignal OUT ab.

Der Vollständigkeit halber sei angemerkt, daß die Registerzelle 55 mit einem Reset-Eingang R und einem Takteingang gekoppelt ist.

Dem im vorstehenden erläuterten Zellenaufbau, d. h. der Zelle 10 ist nun eine Ladelogik 30 übergeordnet, die mit den Multiplexern 56, 57, 58 und 59 verbunden ist und diese den gewünschten Funktionen entsprechend ansteuert.

Sollen zum Beispiel die Signale A2 mit B5 verundet werden, so werden die Multiplexer 56, 57 den Leitungen "ZWEI" beziehungsweise "FÜNF" entsprechend aktiv geschaltet; die Summanden gelangen dann zum UND-Glied 51 und werden bei entsprechender Aktivierung der Multiplexer 58, 59 am Ausgang OUT abgegeben. Soll zum Beispiel eine NAND-Verknüpfung durchgeführt werden, so schaltet der Multiplexer 58 zum Inverter 54 und am Ausgang OUT steht dann das negierte UND-Ergebnis an.

Patentansprüche

1. Datenverarbeitungseinrichtung, wobei ein (im folgenden Datenflußprozessor — DFP — genannter) integrierter Schaltungskreis (Chip) mit einer Vielzahl insbesondere orthogonal zueinander angeordneter homogen strukturierter Zellen mit je einer Mehrzahl jeweils logisch gleicher und strukturell identisch angeordneter Bausteine vorgesehen ist, dessen Zellen zeilen- und spaltenweise, gegebenenfalls gruppenweise zusammengefaßt, mit Ein-/Ausgangsanschlüssen des integrierten Schaltkreises verbunden sind, **dadurch gekennzeichnet,** daß den Zellen eine Ladelogik zugeordnet ist, über die sie je für sich und gegebenenfalls gruppenweise zusammengefaßt so programmierbar (konfigurierbar) sind, daß beliebige logische Funktionen und/oder Vernetzungen untereinander verifizierbar sind, und zwar derart, daß eine Manipulation der DFP-Konfiguration während des Betriebes (oder zur Laufzeit), d. h. die Modifikation funktioneller Teile (MACROS) des DFPs erfolgen kann, ohne daß andere funktionelle Teile angehalten werden müssen oder in ihrer Funktion beeinträchtigt werden.

2. Datenverarbeitungseinrichtung nach Anspruch 1, dadurch gekennzeichnet, daß die Ladelogik mit Speichermitteln gekoppelt ist, über die die Konfiguration der Zellen spezifizierbar ist.

3. Datenverarbeitungseinrichtung nach einem der Ansprüche 1 bis 2, dadurch gekennzeichnet, daß die Ladelogik aus einem Prozessor besteht, der den gesamten Programmablauf auf der Grundlage von in verschiedenen Speichern abgelegten Daten und Programmen im Sinne einer Harvard-Struktur verwaltet.

4. Datenverarbeitungseinrichtung nach Anspruch 3, dadurch gekennzeichnet, daß die Ladelogik ihrerseits aus Zellen mit je einer Mehrzahl jeweils logisch gleicher und strukturell identisch angeordneter Bausteine aufgebaut ist.

5. Datenverarbeitungseinrichtung nach einem der Ansprüche 1 oder 4, dadurch gekennzeichnet, daß die Zellen dynamisch während eines Programmablaufs umkonfigurierbar sind, ohne daß die zu bearbeitenden Daten selbst beeinflußt werden.

6. Datenverarbeitungseinrichtung, wobei ein (im folgenden Datenflußprozessor — DFP — genannter) integrierter Schaltungskreis (Chip) mit einer Vielzahl insbesondere orthogonal zueinander angeordneter homogen strukturierter Zellen mit je einer Mehrzahl jeweils logisch gleicher und strukturell identisch angeordneter Bausteine vorgesehen ist, dessen Zellen zeilen- und spaltenweise, gegebenenfalls gruppenweise zusammengefaßt, mit Ein-/Ausgangsanschlüssen des integrierten Schaltkreises verbunden sind, dadurch gekennzeichnet, daß eine Mehrzahl von ihnen in Kaskadenform koppelbar sind.

7. Datenverarbeitungseinrichtung nach einem der Ansprüche 1 bis 6, gekennzeichnet durch, die Zuordnung geeigneter Daten-Ein-/Ausgabe-Einheiten und mindestens eines Speichers zum Aufbau eines (komplexen, kompletten) Rechenwerks.

8. Datenverarbeitungseinrichtung nach einem der Ansprüche 1 bis 7, dadurch gekennzeichnet, daß die Funktionen der Ein-/Ausgabeeinheiten partiell im integrierten Schaltkreis (Chip) implementierbar sind.

Hierzu 11 Seite(n) Zeichnungen

– Leerseite –

Nummer: DE 44 16 881 A1
Int. Cl.[5]: G 06 F 15/80
Offenlegungstag: 17. November 1994

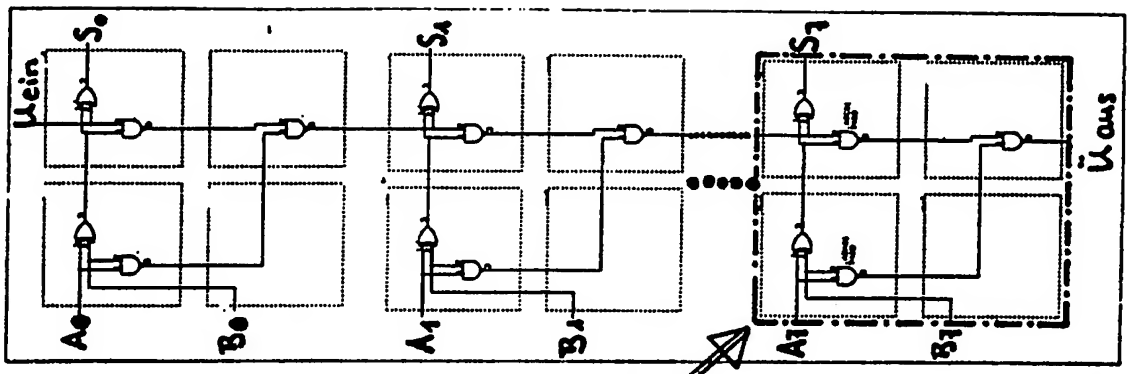


Fig. 5

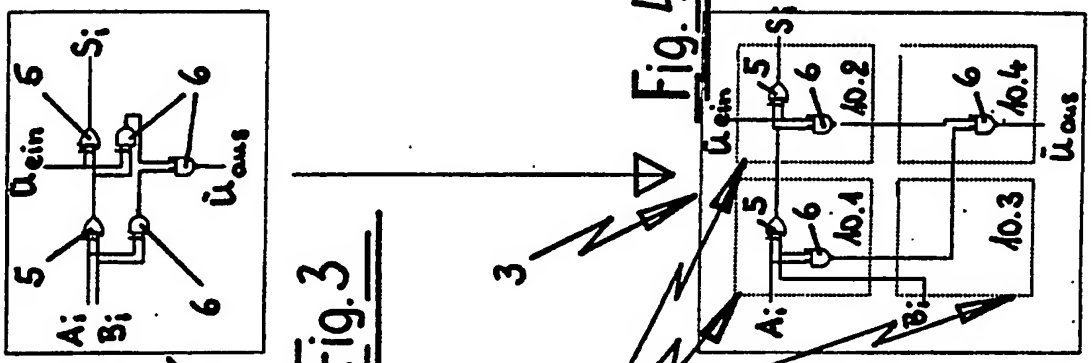


Fig. 4

Fig. 3

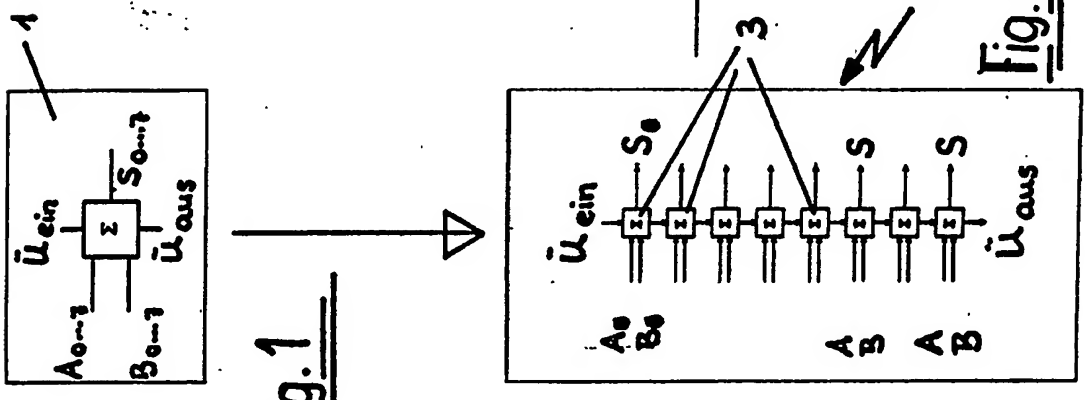
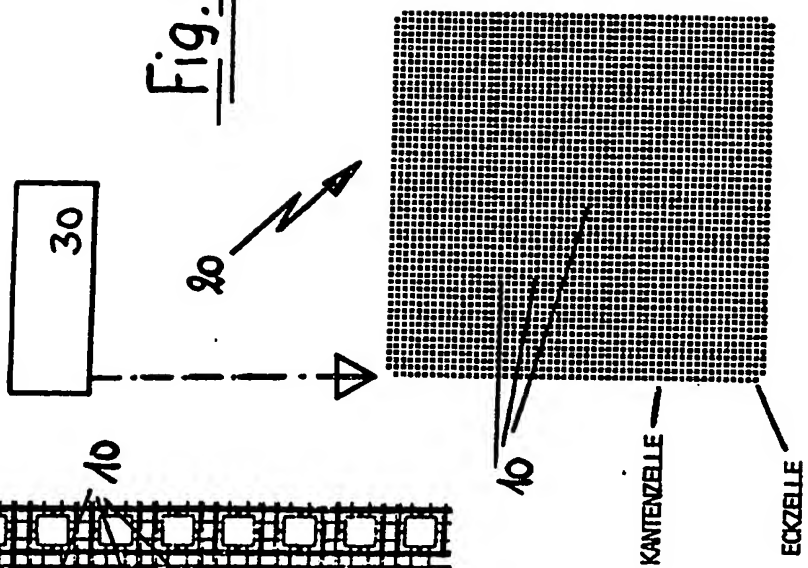


Fig. 2

Fig. 1

Nummer: DE 44 16 881 A1
Int. Cl.[5]: G 06 F 15/80
Offenlegungstag: 17. November 1994

Fig. 8



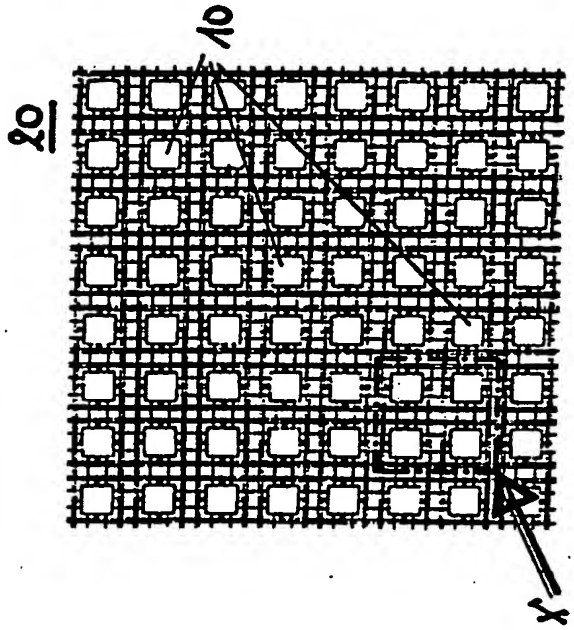


Fig. 7

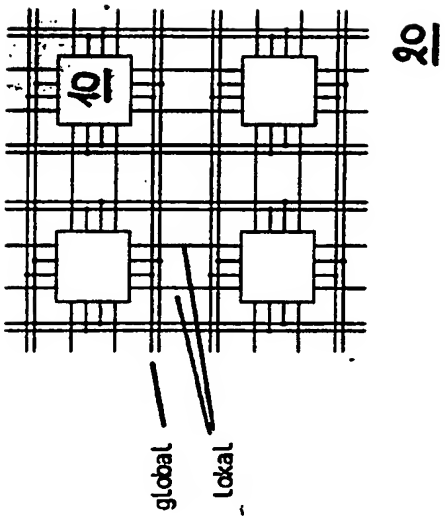


Fig. 6

Nummer: DE 44 16 881 A1
Int. Cl.[5]: G 06 F 15/80
Offenlegungstag: 17. November 1994

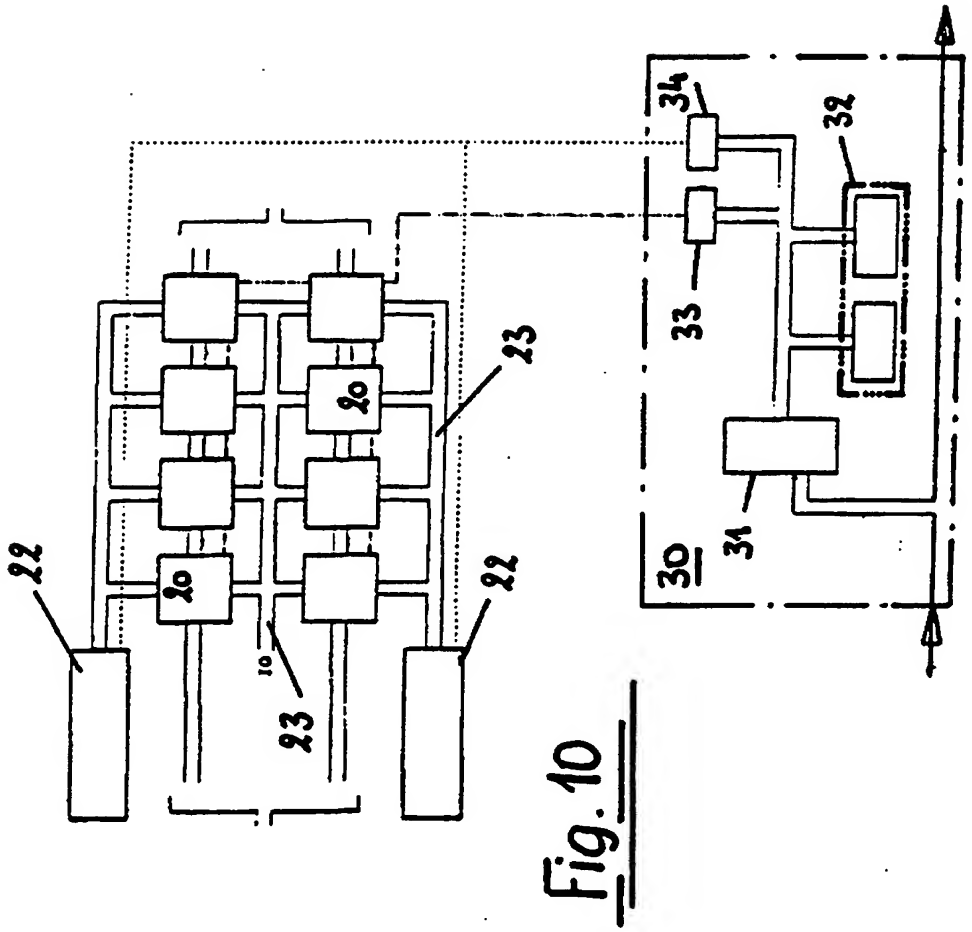


Fig. 10

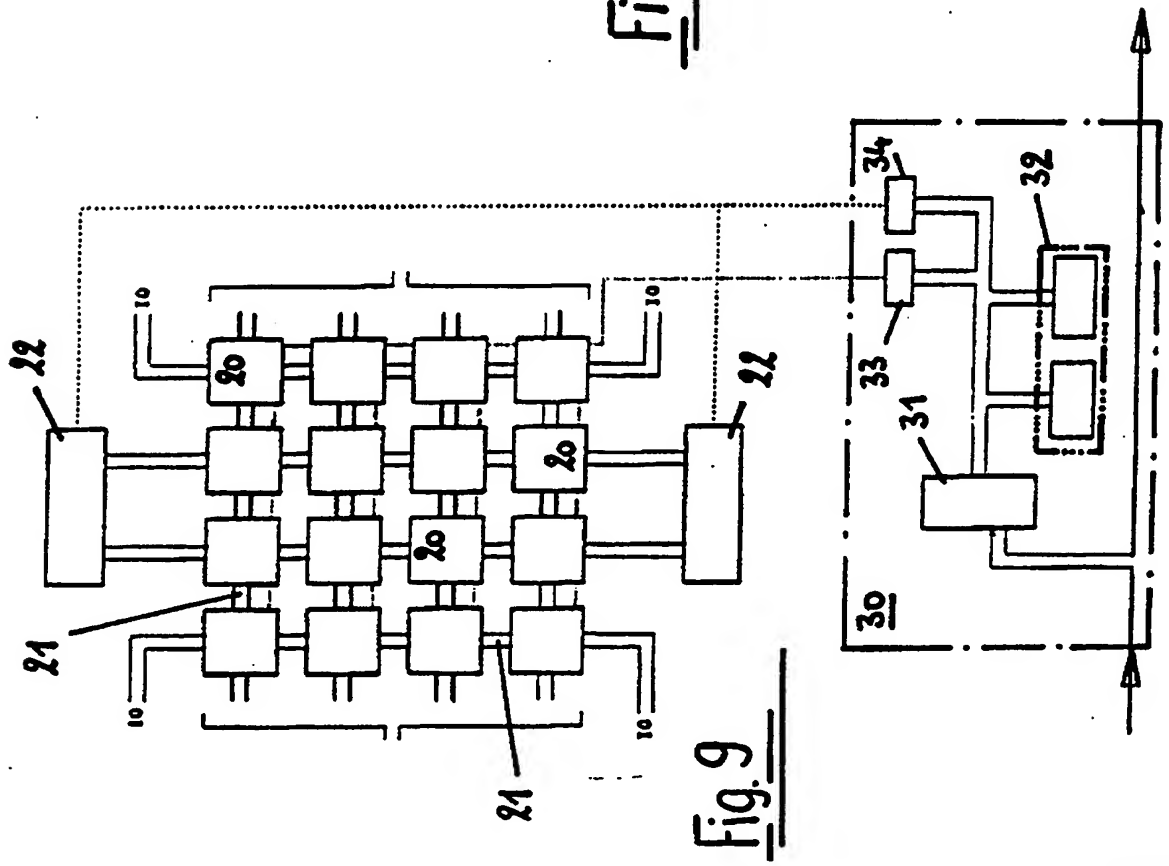


Fig. 9

Nummer: DE 44 16 881 A1
Int. Cl.[5]: G 06 F 15/80
Offenlegungstag: 17. November 1994

Fig. 11 a

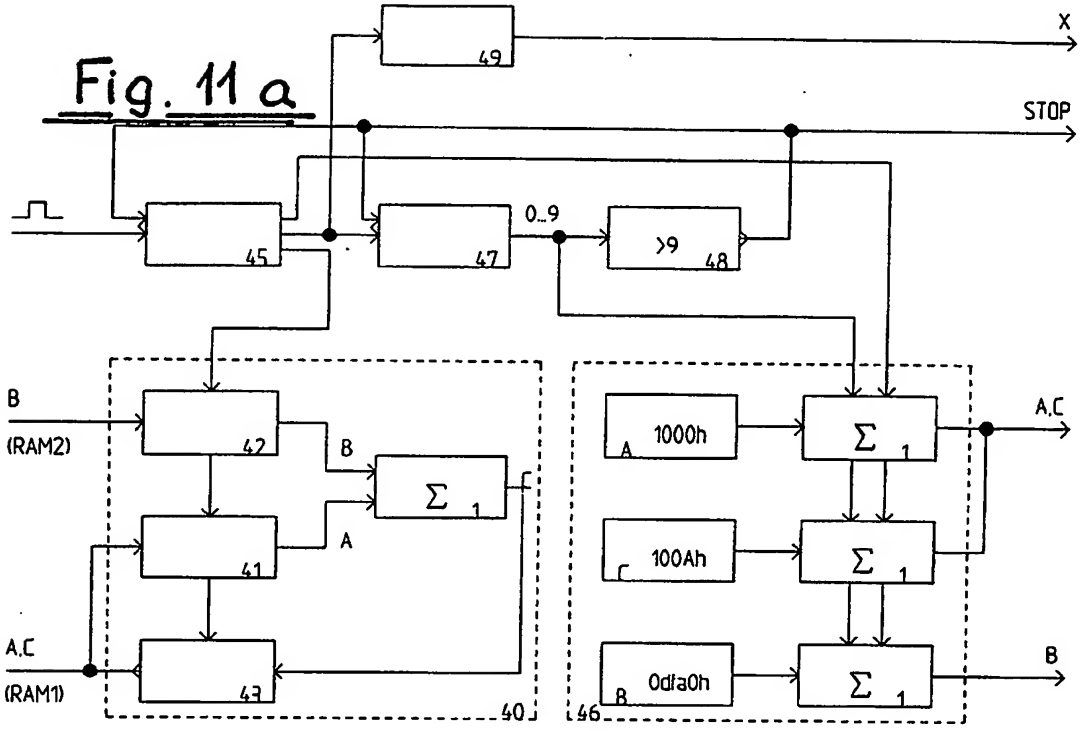
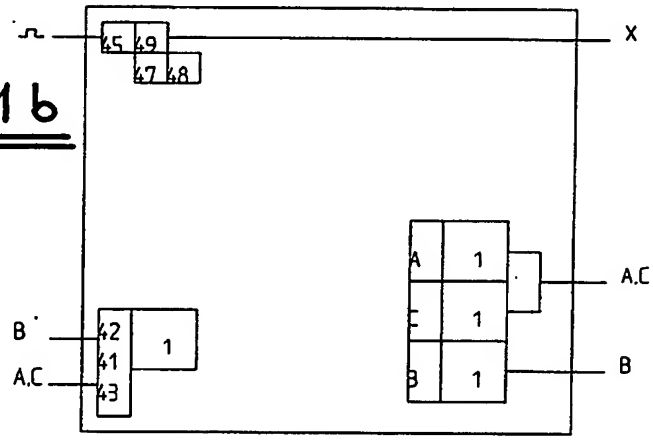


Fig. 11 b



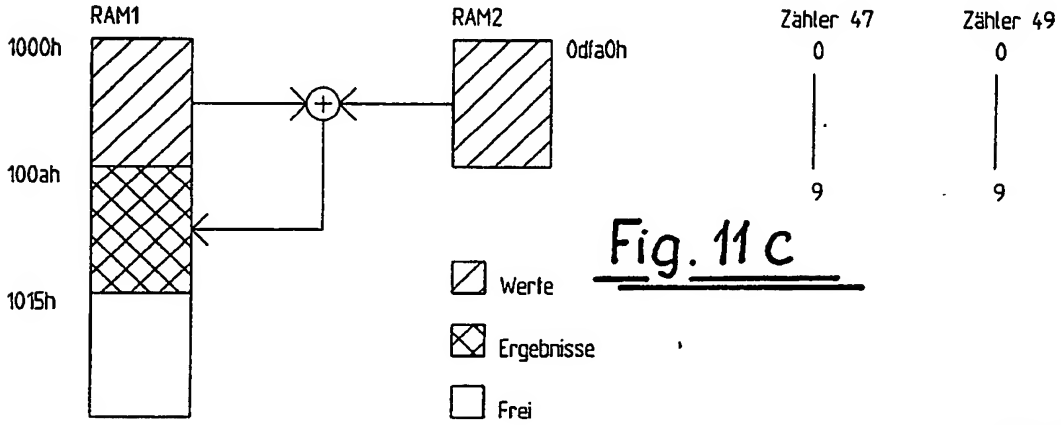
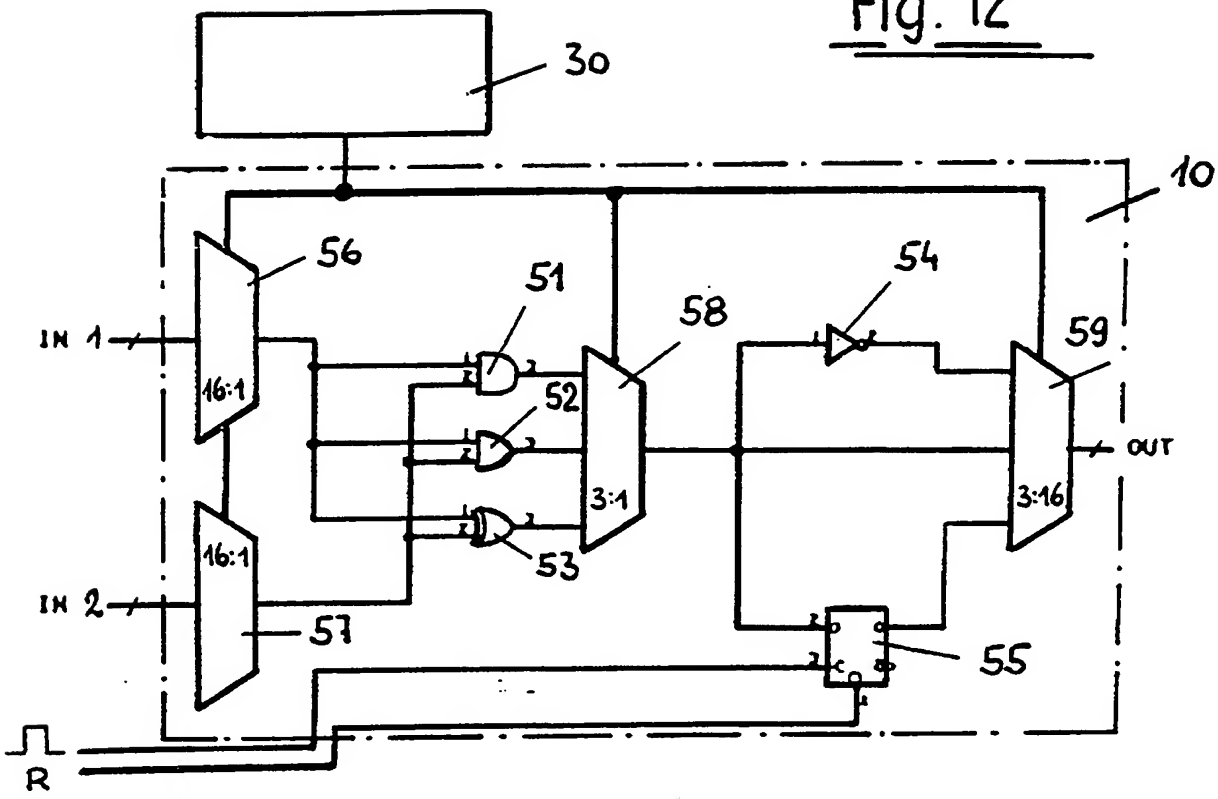


Fig. 11 c

Nummer: DE 44 16 881 A1
Int. Cl.[5]: G 06 F 15/80
Offenlegungstag: 17. November 1994

Fig. 12



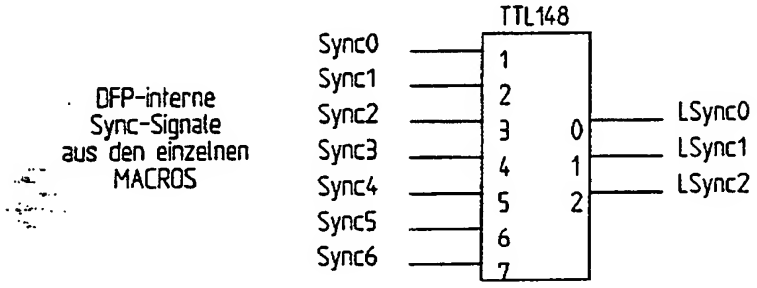


Fig. 13

Nummer: DE 44 16 881 A1
Int. Cl.[5]: G 06 F 15/80
Offenlegungstag: 17. November 1994

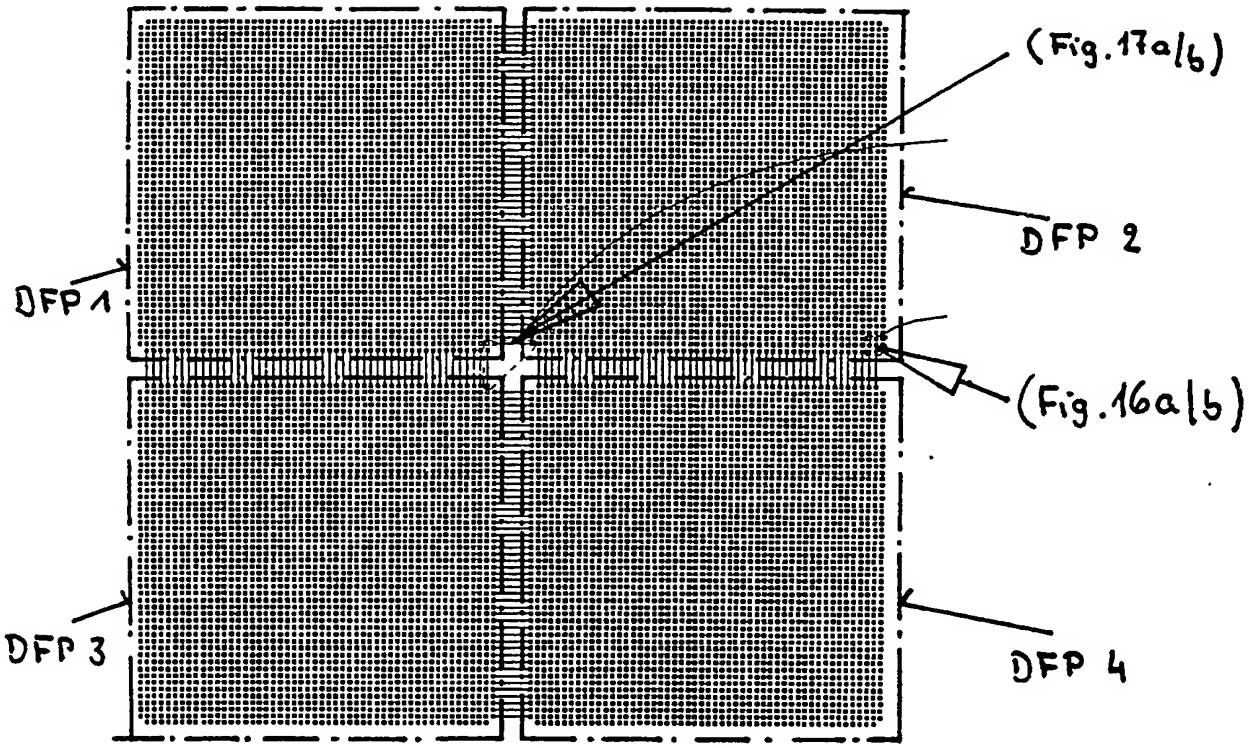


Fig. 14

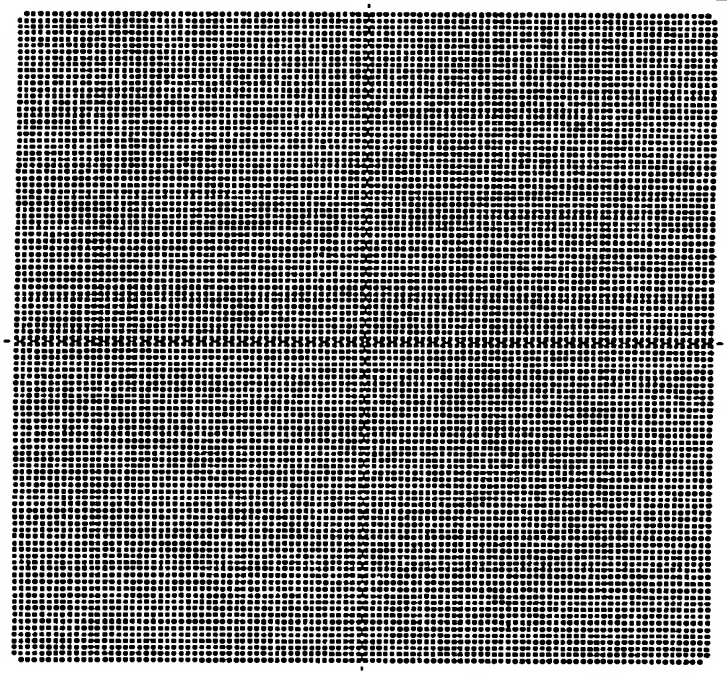


Fig. 15

Nummer: DE 44 16 881 A1
Int. Cl.[5]: G 06 F 15/80
Offenlegungstag: 17. November 1994

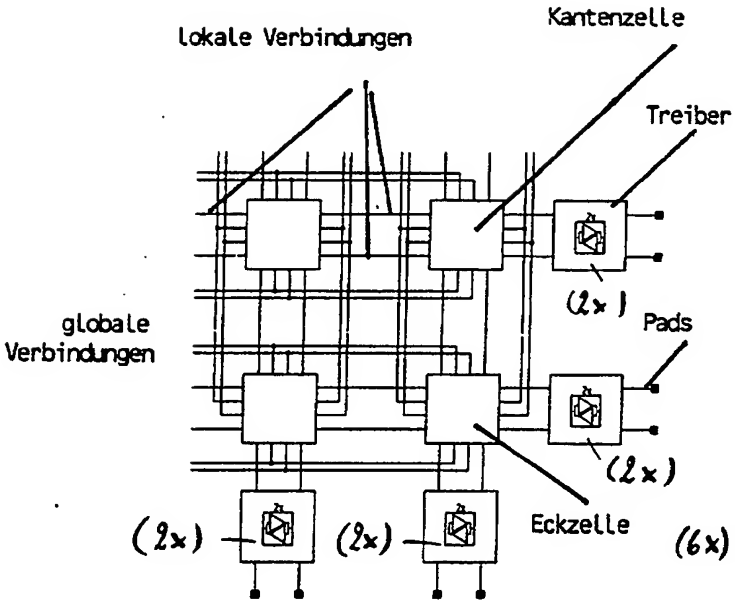


Fig. 16a

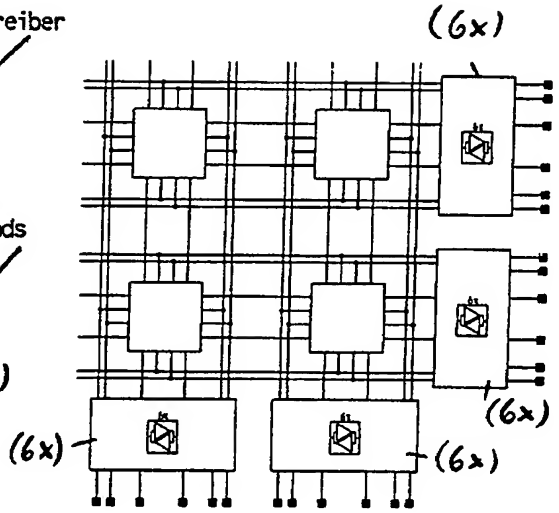


Fig. 16b

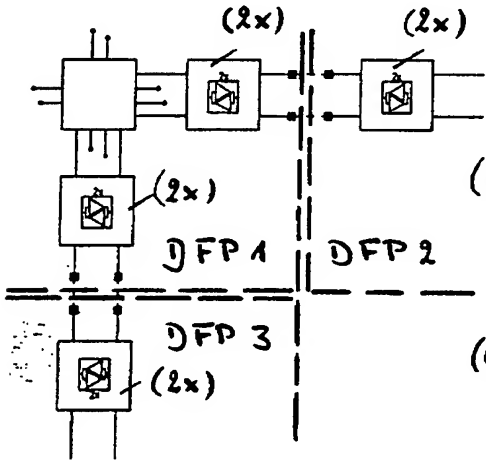


Fig. 17a

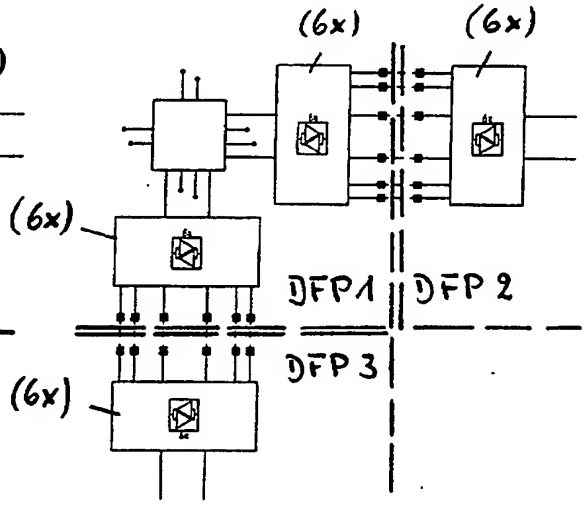


Fig. 17b

Nummer: DE 44 16 881 A1
Int. Cl.[5]: G 06 F 15/80
Offenlegungstag: 17. November 1994

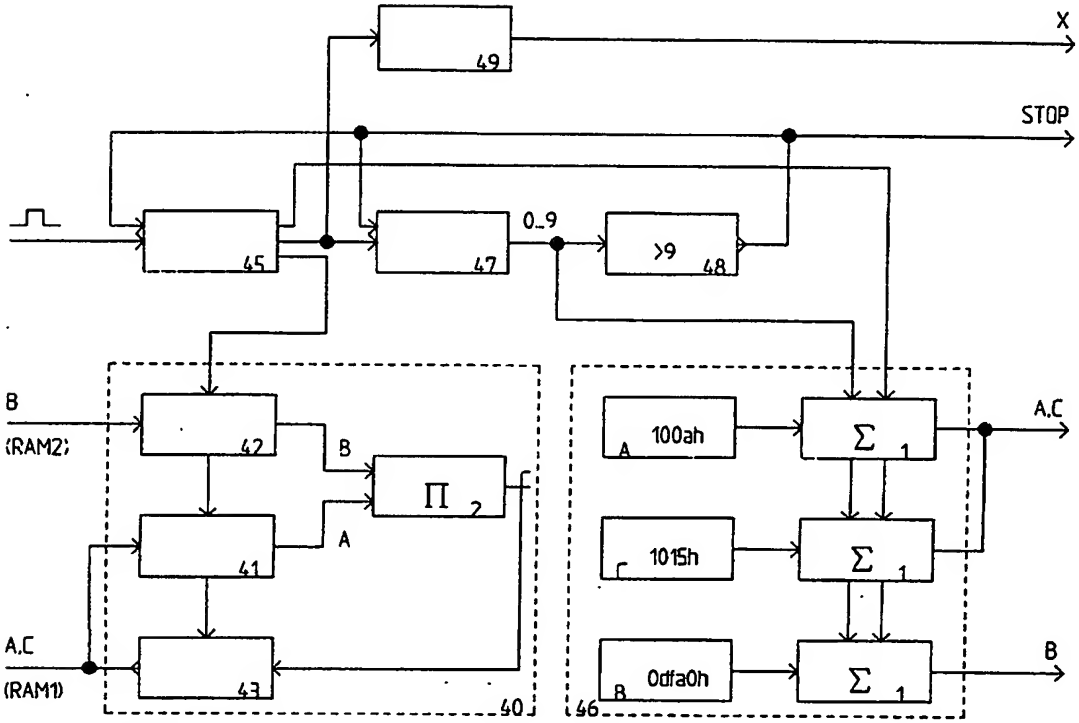


Fig. 18a

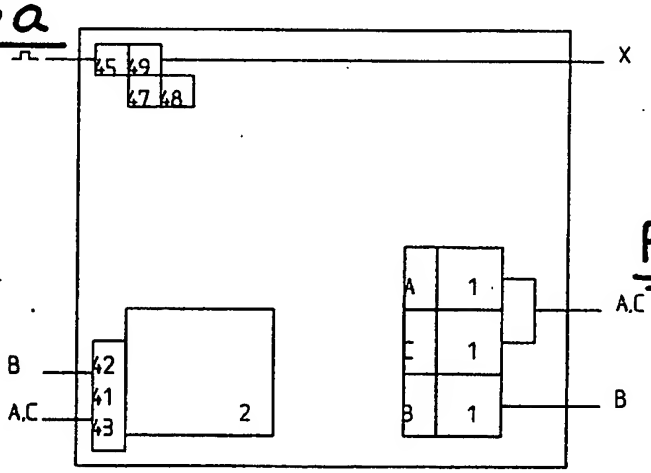
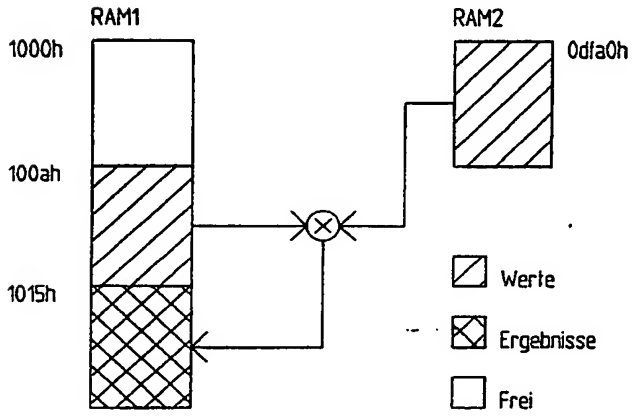


Fig. 18b



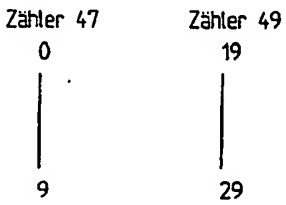
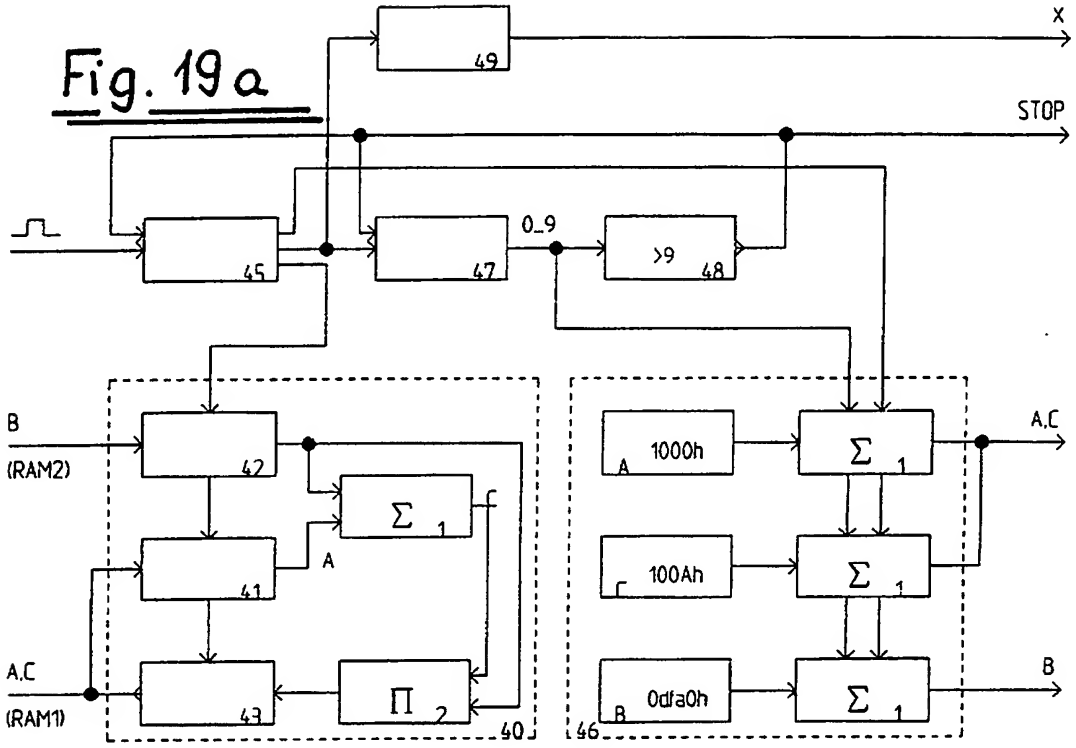


Fig. 18c

Nummer: DE 44 16 881 A1
Int. Cl.[5]: G 06 F 15/80
Offenlegungstag: 17. November 1994

Fig. 19a



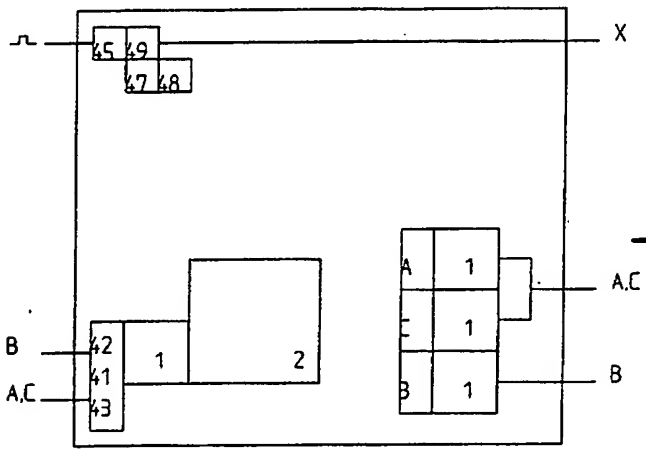
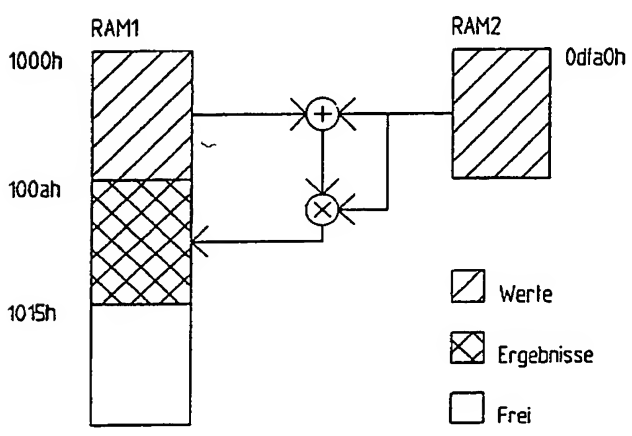


Fig. 19b



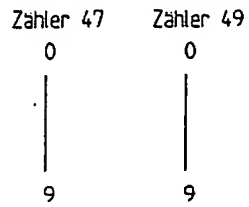


Fig. 19c

Nummer: DE 44 16 881 A1
Int. Cl.[5]: G 06 F 15/80
Offenlegungstag: 17. November 1994

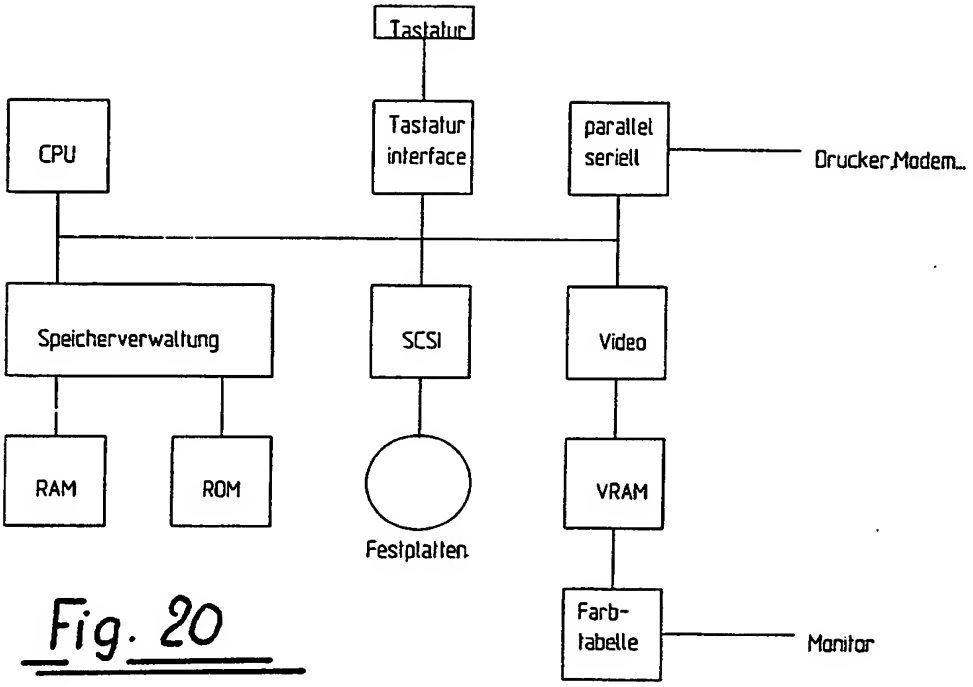


Fig. 20

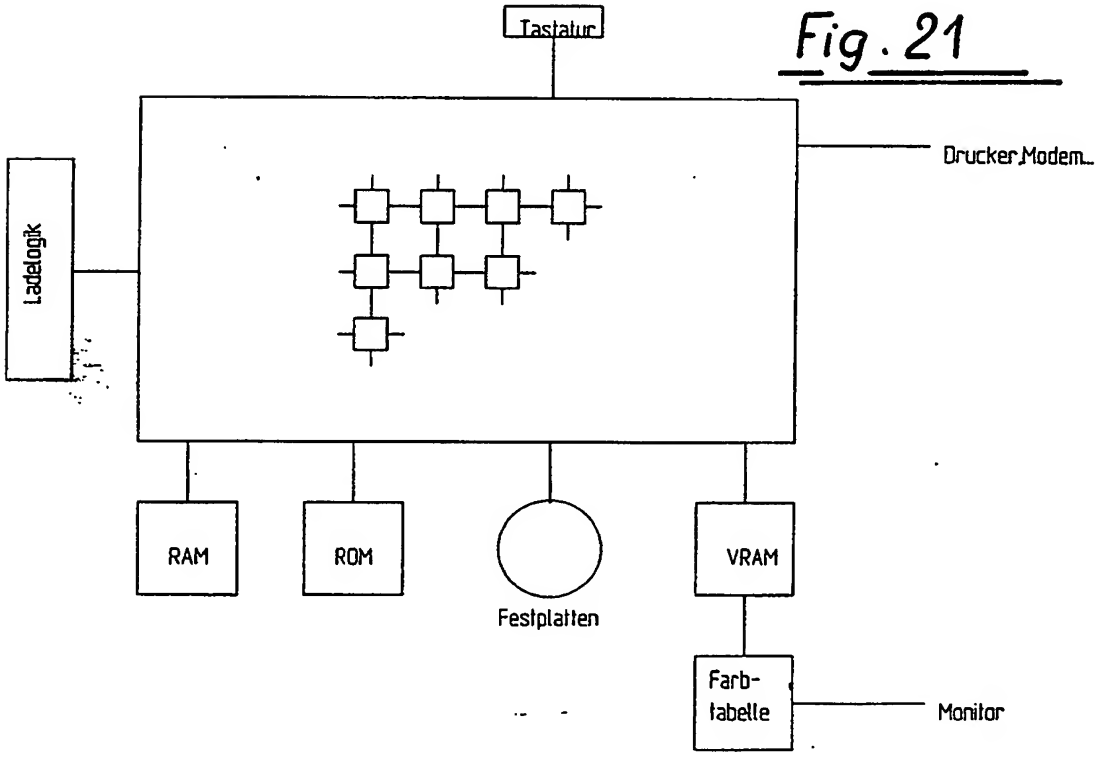


Fig. 21

Nummer: DE 44 16 881 A1
Int. Cl.[5]: G 06 F 15/80
Offenlegungstag: 17. November 1994

Fig. 22